Gruppetime 9

Tråder!

Paralellitet

- Så langt har alle programmene vi har skrevet vært sekvensielle.
- Koden vi skrives kjøres på en kjerne i prosessoren vår.
- Paralellitet lar oss utnytte flere kjerner samtidig.

Tråder

- Thread-objektet i Java kan ta en sekvens instruksjoner
- Disse kan eksekveres på forskjellige kjerner parallelt
- Trådklassen har en konstruktør som tar in en klasse med Runnable-interfacet
- Instruksjonene skrives inn i run()-metoden
- Vi starter tråden med å kalle på .start()

```
@FunctionalInterface
public interface Runnable {

    Runs this operation.

    void run();
}
```

Liten oppgave

Lag din første tråd

- Lag en klasse som implementerer Runnable og printer noe i run()
- I main-programmet, lag et nytt Thread-objekt
- Kall .start() på objektet

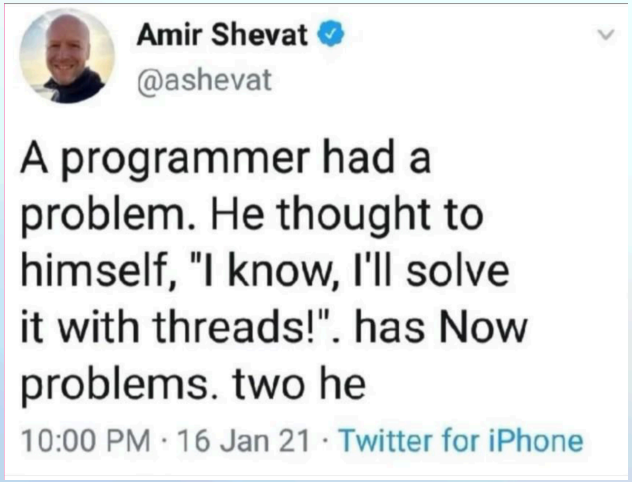
```
@FunctionalInterface
public interface Runnable {

    Runs this operation.

    void run();
}
```

Kjøring av parallell kode

- Man kan ikke garantere rekkefølgen koden eksekveres på
- *Men* man kan vente på den




```java
public class Main {
    public static void main(String[] args) {
        Thread tråd = new Thread(new ParalellKode());
        tråd.start();

        try {
            tråd.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Sekvensiell Kode");
    }
}
```

Hva gjør denne koden?

```
public class Main {
    public static void main(String[] args) {
        Runnable task = new MyTask();
        Thread thread1 = new Thread(task);
        Thread thread2 = new Thread(task);
        thread1.start();
        thread2.start();
    }
}
```

```
class MyTask implements Runnable {
    private final int MAX_COUNT = 10000;
    private int sharedCounter = 0;

    public void run() {
        System.out.println("Starting! Shared counter = " + sharedCounter);
        for (int i = 0; i < MAX_COUNT; i++) {
            sharedCounter = sharedCounter + 1;
        }
        System.out.println("Done! Shared counter = " + sharedCounter);
    }
}
```

Tråder kan overskrive hverandre

- En *ReentrantLock* kan garantere at kun én tråd opererer på variabelen samtidig
- private final Lock lås = new ReentrantLock();
- Kall på lås.lock() før variabelen skrives til, og lås.unlock() etter.
- Unlock() bør skrives i en *finally* klamme for å garantere at verdien alltid blir frigitt

Oppgaver

